SFA Modernization Partner
United States Department of Education
Student Financial Assistance

Integration Application and Technical Architecture Standards

Task Order #4
Deliverable #4.1.5

April 5, 2000

Table of Contents

1 Introduction

This document is intended for use by Management, Systems Administrators, Application Developers, Standards Committees, and any others that will support MQSeries networks or design MQSeries applications. It is designed to provide a common base from which all MQSeries standards can be developed.

Its intended benefits are as follows:

- Consistency in applications and administration processes
- Maximum availability of applications
- Avoiding common mistakes made by beginners
- Assistance to those in the early stages of becoming MQSeries experts
- General assurance of a smooth start for successful MQSeries projects

1.1 Overview

General Items
Provides a general introduction to the MQ Series standards. There are some basic recommendations that apply for any MQSeries implementation. This section covers MQSeries defaults as well as general naming standards.

MQ Network Structure
As the use of MQSeries grows in the technical environment, it is helpful to consider the MQSeries Network Structure separately from the applications. The network structure is the collection of queue managers and the connections between them. These components do not necessarily have any knowledge of the applications they support. They are able to support multiple applications, or run additional applications without modification.

Applications
The general goal behind the application recommendations is to make applications transparent to the MQ network structure. They do depend on MQSeries for message delivery, but the application specific configuration does not have to depend on how that structure is implemented.

2 General Items

A key element of success in using MQSeries is to plan ahead**,** and one important aspect of this consists of adopting a set of workable standards and conventions. The aim of this document is to address these standards and conventions.

This document, which is based on the MQSeries SupportPac MD01, intends to recommend standards and guidelines regarding MQSeries. This will allow the capabilities of MQSeries to be leveraged in the way it was designed.

The recommendations that follow have been built upon previous implementation experience with MQSeries. These standards and guidelines should be augmented by adding additional standards as needed.

2.1 Administration

❑ **Identify the MQSeries Administrator**

Identification of an MQSeries Administrator to monitor the MQSeries environment is critical for adherence to the standards and guidelines identified within the organization. Specifically,

- A single MQSeries Administrator (or a small team dividing the responsibilities for mainframe and distributed platforms) should be part of the IT organizational structure

- The MQSeries Administrator needs to have had appropriate MQSeries training. Ideally, the MQSeries Administrator should be an IBM Certified Specialist in MQSeries.

- The MQSeries Administrator(s) will also need to work in conjunction with security and network administrators.

- Information about MQSeries education and the Certification program can be found on the MQSeries web site.

The key point is that the role is identified as soon as possible.

2.2 Object names

All MQSeries names should follow MQSeries naming conventions, rather than the standard for object names on each supported platform. Key standards and guidelines:

- **Don't use lower case letters**

  MQSeries allows both upper and lower case letters in its names. However, MQSeries names are **case-sensitive**. Using lower and upper case characters for object names is a common source for naming errors.

- **Don't use % in names**

  This character is valid in all MQSeries names, although it is not commonly used in other names across platforms.

- **Choose meaningful names within the constraint of the standard**

  Using meaningful names aids the MQSeries Administrator in maintaining the MQSeries environment

  There is no implied structure, or hierarchy, in an object name, such as you might find on many systems' file names. MQSeries just compares the name strings.

  These standards do recommend using hierarchical names under certain conditions. One such example is to use a suffix where there are multiple "versions" of an object.

- **Document object names and always include a description**

  All objects have a DESCR attribute for this purpose. MQSeries takes no action on the value, but it provides additional information as to the function of the queue.

- **Save the definitions**

  There are a number of reasons for saving the definitions:

  - In the case of a system failure objects may need to be recreated. To perform this function, the definitions need to be saved separately from the queue manager.

  - They can be used to reset the attributes to a known state. For example if triggering has been turned off, or GET or PUT disabled, it is helpful to be able to restore the objects to their initial state.

  - The definitions can supplement the MQSeries documentation.

2.3 Defaults

- **Ideally leave defaults unchanged**

  MQSeries generally keeps attribute defaults in standard objects, 'SYSTEM.DEFAULT.*'. When an object is defined, MQ takes any unspecified attributes from the corresponding default object.

  The original intent of this approach was to support users who wanted to have different defaults. The various platforms supply these defaults in different ways.

  - MQSeries for MVS/ESA provides a script which can serve as the "Initialization Input Data Set" in the queue manager JCL.
  - MQ Series Version 5 (for NT and UNIX) implementations create the standard default objects automatically when a queue manager is created.

  A general rule of thumb is to accept MQSeries defaults unless there is a good reason to change them.

- **If you must change defaults, use a Customization file**

  Don't change the supplied script, changes are lost if there is a subsequent product update.

  The recommended approach is to have a separate Customization file and use **ALTER** commands to set a new value for the identified attribute

- **Use the customization file for Queue Manager attributes**

  Some queue characteristics are configured when a queue manager is created, and can not be changed after the initial configuration.

  Specifying queue manager attributes in a Customization file, in addition to being easier to manage, provides a direct way for all the values to be returned to a known state.

  - **Example 1**, on an MVS queue manager:

```
ALTER QMGR  +
    DESCR('Queue manager = MARS')   +
    DEADQ('SYSTEM.DEAD.LETTER.QUEUE')
```

  - **Example 2**, on one of several UNIX systems connected to that MVS queue

```
ALTER QMGR  +
    DESCR('Queue manager = JUPITER4')   +
    DEADQ('SYSTEM.DEAD.LETTER.QUEUE')  +
```

manager:

- **Use templates for default classes**

  Objects are defined with reference to a known Defaults object (a template object). Identify these clearly by using “TEMPLATE” as part of the name.

3 MQ Network Structure

3.1 Queue managers

- **Assign unique names to all queue managers**

This recommendation can often cause significant problems if queue manager names are not unique. (On MVS, the queue manager name must also be distinct from other subsystem names on the same system.)

A queue manager can be understood as a "container" for queues and related objects. There is typically one per system, but additional queue managers can be defined.

Queue Managers with the same name can be configured to exchange messages - by using Queue Manager aliases. This is strongly discouraged. There are some examples where this can lead to ambiguity, and messages can then be sent to the wrong queue manager.

  - If ReplyToQMgr is left blank in the Message Descriptor, MQSeries inserts the actual local Queue Manager name, not its alias.
  - Dead Letter Queue messages identify the real Queue Manager, not any alias.

- **Don't copy documentation examples**

Copying the documentation examples provided with the installation files is an easy way to produce queue managers with duplicate names. Plan for the names of queue managers ahead of time.

- **Keep the queue manager name short and meaningful**

A recommendation would be to make queue manager names the same as the network host name. However, keep the following points in mind:

  - On MVS, the queue name has to be the same as the host name. The queue manager name corresponds to the MVS subsystem name. Therefore, the queue manager name is restricted to **4 characters**.
  - Many queue managers use the **first 8 characters** when generating unique message identifiers.

- Channel names, which by convention are derived from queue manager names, are limited to 20 characters.

If there is no obvious name, most users would adopt a convention for constructing a queue manager name. Make sure that the convention provides for further expansion, particularly where the restricted names on MVS are concerned.

Some naming examples are illustrated below. A numeric identifier may be appropriate where a processor (or hardware cluster) has multiple queue managers.

- Example: CCCDDFNN
  CCC = city identifier
  DD = company division
  F = queue manager function (e.g. Test)
  NN = numeric identifier

- Example: SSSCCFNN
  SSS = stock ticker symbol
  CC = city identifier
  F = queue manager function
  NN = numeric identifier

- MVS Example: ADDX
  A = geographic area
  DD = company division
  X = distinguishing identifier

- **For a Queue Manager alias, add a suffix to the name**

The main use for this would be to support **classes of service**. There are fewer constraints on the length of an alias name; it can be more than eight (or four on MVS) characters for example.

In fact this feature is usually related to defining multiple channels between a pair of queue managers. In this case, use the same suffix for associated channels and queue manager aliases.

- **Example 3**, the UNIX queue manager in Example 2 needs an alias so it can receive very large reply messages on a separate channel.

```
DEFINE QREMOTE('JUPITER4_XL')  REPLACE  +
    DESCR('Queue manager alias for very big messages')
```

3.1.1 Default Queue Manager

- ❑ **Don’t identify any single Queue Manager as the default**

  Some environments can tolerate an exception, most notably CICS/ESA, where any CICS region is always connected to a single Queue Manager.

  Most platforms can have more than one queue manager defined on a system. Don't pick one as the default, as this often results in selecting the wrong queue manager on a particular system.

  Even when there is only one queue manager configured, don't define it as the default. Doing so increases the probability for error if another queue manager should be added at a later date.

- ❑ **Pass the connection name as program parameter**

  This allows a program to run unchanged on any Queue Manager. This provides the capability for multiple concurrent instances; or a queue driven application could be moved to a different queue manager without impacting the application code.

3.2 Dead Letter Queue

If MQSeries can detect an error synchronously, it is reported directly to the application; if a message can not be delivered synchronously, it is a candidate for the Dead Letter Queue. The Dead Letter Queue preserves a message that can not be processed immediately without stopping valid messages in the meantime.

- ❑ **Include a Dead Letter Queue on all queue managers**

  On all queue managers, use a local queue called SYSTEM.DEAD.LETTER.QUEUE.

  This queue is created automatically on some MQSeries platforms. On those platforms where the queue is not automatically generated, create a queue with this same name. It will cause less confusion to use a common name for the dead letter queue across all platforms.

  Note: It is still necessary to configure the queue manager, by identifying this queue in its DEADQ attribute.

  If a Dead Letter Queue is required, and is not available, a channel will fail.

- **Process undelivered messages**

  Messages that are put on the Dead Letter Queue take the form of the original message data, preceded by a **dead letter header** - defined by the MQDLH structure. The header includes the intended destination queue and queue manager for the message, and the reason the message could not be delivered.

  The production environment must have a process (triggered or scheduled at intervals) to dispose of undelivered messages in the Dead Letter Queue. Some platforms supply a Dead Letter Queue Handler (a rules driven application to manage dead letter messages). Other platforms will require development of a program for this purpose. To handle the dead letters:

  - Construct rules based on queue names, message type, feedback code, etc. It can be appropriate in some cases to retry or discard certain messages

  - Otherwise, transfer the undelivered message to an application queue to allow the application to process the message

3.3 Queue Manager Clusters

This facility allows a name to be given to a collection of queue managers, and was introduced in MQSeries Version 5 for AIX, HP-UX, OS/2, Sun Solaris and Windows NT; and MQSeries for OS/390 Version 2.1. It simplifies administration by providing a single system image, and it supports dynamic workload balancing.

- **Choose a unique cluster name that describes its function**

  Cluster names are not restricted to short names (like queue managers), nor is it required to state in the name that it is a cluster. The recommendation is to name clusters with a descriptive name, and add a suffix if there is a requirement for multiple overlapping clusters.

3.4 Channels

MQSeries automatically defines the internal channel connections and transmission queue within a cluster. However, transmission queues and internal channel connection names can be modified after the channels are created.

3.4.1 Transmission queue

- **Use exactly the same name as the destination queue manager**
  MQSeries will select the transmission queue name in the absence of other information. Note you can not rely on there being a QREMOTE attribute to define a transmission queue in all cases. An example of this is a message to a Reply Queue, which will only have a destination Queue Manager name from which to determine routing of the message.

- **Example 4**, the UNIX queue manager in Example 2 needs a transmission queue to access the MVS hub queue manager.

```
DEFINE QLOCAL('MARS')  REPLACE  +
     DESCR('Transmission queue, sending to MARS')    +
     USAGE(XMITQ)  TRIGGER           +
     INITQ('SYSTEM.CHANNEL.INITQ')   +
     TRIGDATA('MARS')
```

- **If there is more than one channel, add a suffix**

  The recommended standard is to specify your queue manager alias as the ReplyToQMgr. The remote system will use that attribute as the transmission queue for its reply.

  Use the same suffix for a transmission queue and its destination Queue Manager alias

  - **Example 5, the same UNIX queue manager has a separate channel to send** large messages.

```
DEFINE QLOCAL('MARS_XL')  REPLACE  +
     DESCR('Transmission queue, big messages to MARS')   +
     USAGE(XMITQ)  TRIGGER           +
     INITQ('SYSTEM.CHANNEL.INITQ')   +
     TRIGDATA('MARS_XL')
```

- **Be careful using the default transmission queue**

  This feature is not available on all platforms. On platforms where it is supported, it is a convenient way to avoid having to define a transmission queue (and channel) for all possible destinations.

  Default transmission queues are particularly useful for end point nodes in an MQSeries network.

  A configuration to be aware of and avoided is a loop of default transmission queues. MQSeries does not detect this situation, and continues to forward the messages in an infinite loop.

- **Make triggering standard for a Sender channel**

  Configure all transmission queues for triggering. To do this the Channel Initiator will need to be started. Triggering standards requirements include:

  - Always use trigger type FIRST, and TRIGMPRI(0).
  - On MQSeries Version 5 (NT and UNIX) platforms, the corresponding channel name is specified as Trigger Data. Configure Process object as instructed in the installation documentation.
  - Use the supplied Initiation Queue name, 'SYSTEM.CHANNEL.INITQ'

  A Requester channel is intended to initiate message transfer from the destination system. As a result, the corresponding Server channel does not need to be triggered.

3.4.2 Message channels

- **Name a channel the same as the <u>destination queue manager</u>**

  This means a queue manager only needs one Receiver channel defined, no matter how many queue managers will send messages to it. Each adjacent source queue manager needs a Sender channel and a transmission queue – all matching the name of the destination queue manager.

  This scenario can be generalized to multiple channels, where the transmission queue and channel would match the receiving queue manager alias – i.e., they all have the same suffix.

  The same convention applies to dynamic channels, introduced in MQSeries Version 5. If a sender channel is started, and the corresponding Receiver channel has not been defined, the Receiver is created automatically.

- **Include the transport type if it adds value**

  Where queue manager are found in a mixed network, is it helpful to indicate the network protocol in the naming convention. If this is needed, make the transport distinction evident in the **class of service suffix**; for example 'JUPITER4**_SNA**'.

3.4.3 Client Connections

- **Don’t create a channel for each separate client**

Defining a separate channel for each client represents unnecessary effort.

Use the same name, '**CLIENTS**', on all queue managers. If multiple connections have to be configured, such as different transport types, add a suffix to this name.

4 Applications

These recommendations assumes an MQSeries infrastructure is in place, such as that described in the previous section. The goal here is to make application code transparent to any configuration changes.

4.1 Queues

4.1.1 Names

- **Name a queue to describe its function**
  A message driven application provides some service. In addition, exclude unrelated information from the name.

- **Use hierarchical names for application queues**
  The form that is often recommended is as follows.

  <application>.<function>

  MQSeries uses the prefix 'SYSTEM.*' for objects it delivers. It is recommended that this prefix is not used for application-related queues.

  Using a prefix to group related queues simplifies MQSeries administration in the following areas:

    - inquiries about queues
    - MVS security administration
    - Dead Letter Queue handler

  In a larger application, it can be appropriate to adopt a naming hierarchy. For example,

  <system>.<application>.<function>.<sub-function>

- **Don't include the Queue Manager name**
  MQSeries generally identifies a queue by a pair of names, the queue name itself and the containing queue manager. Including the queue manager as part of a queue name is redundant and may lead to additional queue management complexities.

Where an application is rolled out over multiple nodes there is no need to invent a new queue name for each instance.

- **Don't include the queue type in the name**

  MQSeries administration makes queue types transparent to applications. The queue type should not be visible in the queue name. If the queue type is changed, then the queue name does not have to be changed as well.

- **Pass the name of the input queue by parameter**

  Each application needs a QLOCAL (local queue) to provide its input. Generalize the application code by passing the queue name as parameter. Multiple instances of an application can use different local queues, without having to change the code.

  Note that programs that are triggered will follow this standard by default—the local queue name is part of the trigger parameter.

4.1.2 Versions

- **Indicate queue versions by a suffix on a local queue**

  There may be occasions when multiple versions of a queue exist at the same time. Reasons for multiple versions include different versions of the function driven by the queue or the application may assign a different local queue based on the time of day.

  Indicate the version in the form of a suffix on a local queue name. Examples include:

  <application>.<function>_TEST
  <application>.<function>_V2.1
  <application>.<function>_THURSDAY

  Using the queue name as a parameter will ensure transparency to the application code.

- **Use a queue alias to PUT messages to the right queue version**

  Using queue aliases is particularly useful when a message is PUT to a queue to request a service. The choice of the correct version of the local queue should not be the responsibility of the requesting program.

  Use the same queue name across all platforms to PUT messages.

Define the queue alias as QALIAS or QREMOTE as appropriate. Do not include the queue type in the name.

If you have a Directory service, use a QALIAS with SCOPE(CELL) instead.

4.1.3 Reply queue

- **The recommended naming convention is <application>.REPLY**

  This fits in with the hierarchy convention described in previous sections.

  Do not include the queue type (QM or QL) in the reply queue name. This is an aspect of the configuration that could vary, such as for performance tuning.

  There are various application approaches to processing a reply queue which imply different queue types.

4.1.4 Dynamic Queues

When MQSeries creates a dynamic queue, the first part of the resulting queue name can be controlled through the Object Descriptor. The appropriate name standard depends on the type of dynamic queue created.

- **Temporary Queues – accept the MQSeries default**

  The MQSeries default for a dynamic queue prefix is 'CSQ.*' on MVS and 'AMQ.*' on other systems. Since temporary dynamic queues are deleted on MQCLOSE, they will not have to be controlled by the MQSeries Administrator so it is recommended that the default is left unchanged.

- **Permanent Queues – supply an application prefix**

  A permanent dynamic queue can remain across application invocations. It may need to be managed by an MQSeries Administrator, so ensure the queue name follows the hierarchical naming convention. Specify an application prefix in MQOD.DynamicQName, followed by an asterisk.

  Note that this application prefix must not exceed 32 characters, so that MQSeries can generate a unique name with the remaining characters.

4.1.5 Namelists

Originally only a function of MQSeries for MVS/ESA, it is now available across the Version 5.1 (NT and UNIX) platforms. With the addition of distribution list support on OS/390, namelists are now a cross-platform facility to maintain a list of queues in MQSeries storage, and for programs to send a message to all queues with one MQPUT.

- ❑ **Use a hierarchical naming convention**
  Don't indicate in the name that it is a Namelist. Namelists have a separate name space , so the identification of a namelist is completely clear from the context of the message.

4.1.6 MQSeries Integrator

MQSeries has now been enhanced with enterprise application integration (EAI) functionality. MQSeries Integrator supplies rules-driven routing and data transformation , which simplifies the task of integrating diverse applications across the enterprise. MQSeries Publish and Subscribe supports routing of topic-based messages to dynamic subscribers based upon the content of the message. These two facilities are compatible, and can be used to construct complex messages and routing based on business logic.

The same general principles can also be used when naming queues associated with these functions:

- Subscriber queues are in fact application input queues, so application naming standards apply to these queues.
- MQSeries Integrator has input queues, which can be given a hierarchical name – just as if the EAI tool was an application, and provides the first part of the queue name.

4.1.7 Queues for Bridges and Links

- ❑ **Include the bridge or link type in the application hierarchy**

For example,

```
<application>.MVS
<application>.CICS
<application>.ORCL
```

4.2 Triggering

Triggering in not required in all cases. Programs can be scheduled based on demand, time of day, or as part of the system startup procedures.

4.2.1 Process

- **If a queue has its own Process, use the same name as the queue**

  Include the version suffix of the queue if appropriate. There may be a separate executable process for each instance of the queue.

  Note that Processes have an independent name space. As a result, there is no value including the fact it is a PROCESS as part of the name.

- **If a Process is shared, describe the collective function**

  Where several queues are handled by a common program, define a single Process object. Use a suitable hierarchical name for the collective function.

  If multiple versions of a queue are read by the same program, drop the version suffix from the queue name.

- **Use Environment Data as a parameter to the trigger monitor**

  It is especially important if the Trigger Monitors are custom developed.

  User Data is intended to be used as parameter information to the triggered program. Trigger Data also provides parameter information that is specific to one queue.

  Some supplied trigger monitors do not use this information. On UNIX, a value of '&' causes the program to be triggered as an asynchronous process.

4.2.2 Initiation Queue

- **Use system defined queues for simple general triggering**

  Some platforms define standard initiation queues when a queue manager is created. Below are the defaults for trigger monitors:

  SYSTEM.DEFAULT.INITIATION.QUEUE
  SYSTEM.CICS.INITIATION.QUEUE

- **Otherwise use a hierarchical name**
  A reasonable approach may be to have an initiation queue for the various functions in an application. The recommendation is to use a queue name of the form,

  <application>.INITQ

- **Hint – to stop any trigger monitor, disable GET for its INITQ**
  Trigger monitors are designed to run for long periods of time. They stop only when MQSeries is shut down, or when the trigger monitor task is cancelled by an operator.

  MQSeries for MVS/ESA provides an interface to stop its CICS Task Initiator function without disrupting other operations.

4.2.3 Trigger control

- **For temporary disabling triggers, use the NOTRIGGER parameter**
  The function of this parameter is to suspend triggering temporarily in an application. Use trigger type NONE for a queue that will never be triggered.

- **Avoid trigger type DEPTH**
  The original intent of this feature was to support consolidation of replies to related parallel requests. The reply queue for the set of related messages would be a permanent dynamic queue, triggered when all the replies had arrived.

  The problem with this type of triggering is that triggering is disabled after the trigger occurs. There is no automatic re-triggering if all the messages are not processed.

  Never use trigger type DEPTH to monitor a queue threshold. Use Performance Event messages.

- **Avoid trigger type EVERY**
  Problems occur with this trigger type when a system is re-started and there are several messages recovered on a queue. Only one trigger will be initiated, no matter how many messages are on the queue. The preferred approach is to use trigger type FIRST, and modify the application to continue processing additional messages.

5 Additional information

5.1.1 References

- **MQSeries home page,** http://www.software.ibm.com/ts/mqseries/
- **SupportPac MD01: MQSeries – Standards and conventions,** http://www.software.ibm.com/ts/mqseries/txppacs/md01.html
- **MQSeries Planning Guide, GC33-1349**
- **MQSeries Intercommunication, SC33-1872**
- **MQSeries Command Reference, SC33-1369**
- **MQSeries Queue Manager Clusters, SC34-5349**